FLAPP-IA BIRD

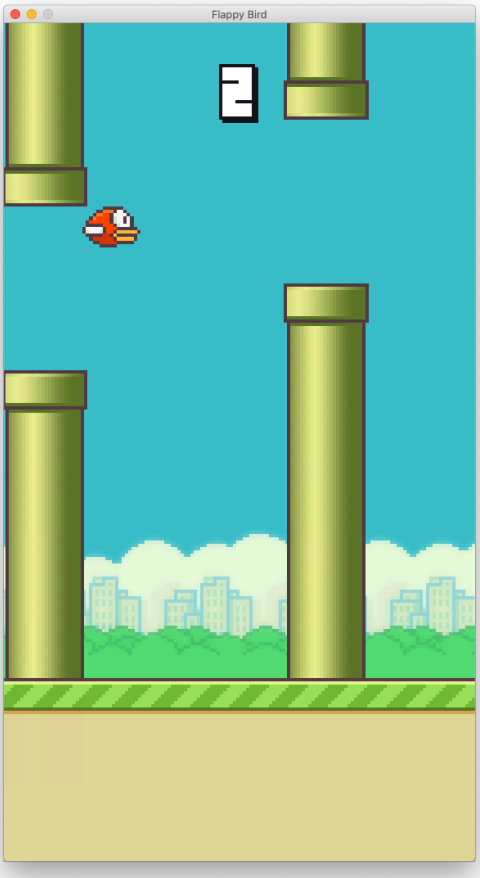
I. Description

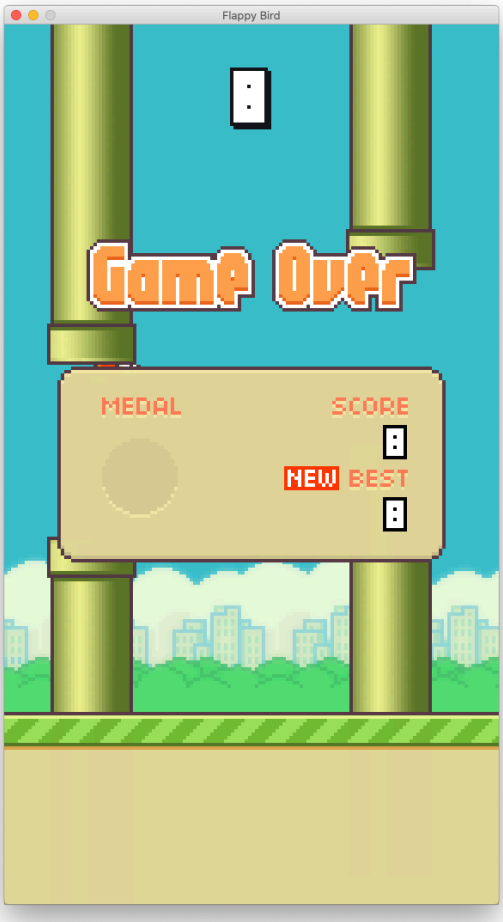
Flapp-IA Bird est une copie du célèbre jeu « FlappyBird » en Python avec un algorithme génétique implémenté qui permet au jeu de se jouer tout seul

A. Fonctionnalités

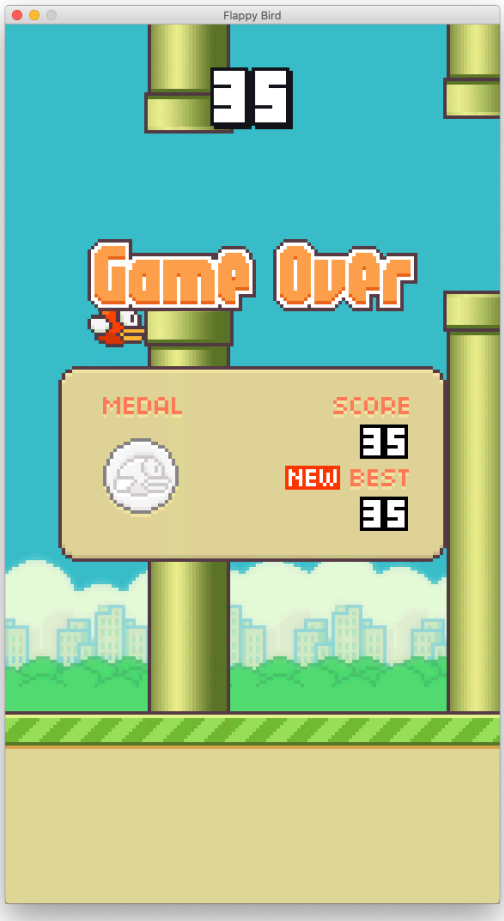
Le jeu possède 2 lancements différents: un mode normal ou l'on peut jouer au jeu, et un autre mode ou l'on observe le jeu apprendre à jouer tout seul.

La version jouable comporte un écran d'accueil, et une copie conforme du jeu original

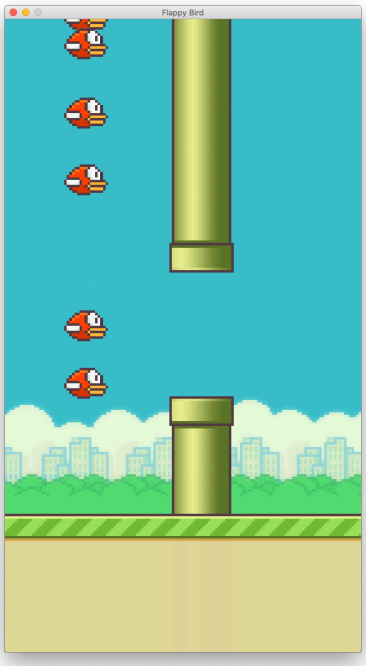




À la fin de la partie, le jeu affiche votre score, si c'est un meilleur score, il l'enregistre et indique que votre score actuel est votre nouveau record.



Aussi, comme le vrai jeu, celui-ci affiche votre « médaille » selon votre score (bronze, argent, or, platine)



L'autre mode est donc le mode où l'intelligence artificielle génère des oiseaux qui jouent au jeu et se reproduisent selon leurs performances pour devenir de plus en plus fort

B. Technologies

Ce jeu a été codé en Python, l'interface graphique est réalisée à l'aide de la librairie Pygame et l'algorithme génétique a pu être grandement simplifié grâce à la librairie NEAT.

II. Le jeu

A. Les classes

Le jeu comporte 5 classes:

- La classe Game : le jeu principal, qui comporte l'ensemble des données du jeu, ainsi que la boucle principale du jeu

- La classe Bird : Permet de faire des instances d'oiseaux, comporte les coordonnées de l'oiseau et les fonctions qui permet au joueur de sauter, d'y affecter la gravité et gérer les contacts avec les Tuyaux

- La classe Tuyau : Permet de crée un obstacle, avec une hauteur aléatoire. La classe Game possède une liste de tuyaux à afficher à l'écran

- La classe Score : Permet de gérer le score de l'utilisateur, le meilleur score, les médailles et gère l'affichage à l'écran de ceux-ci

- La classe Ground : Classe anecdotique qui sert uniquement à animer le sol pour donner une illusion à l'oiseau d'avancer

B. Les fonctions majeures

La boucle game comprend entre autre la fonctionnalité de faire sauter l'oiseau:

```
if keys[pygame.K_SPACE] and self.launch: # Le joueur appuie sur espace et la partie est lancée
    self.bird.jump() # Utilise la méthode "Jump" de l'instance bird du jeu
```

Ainsi dans la classe Bird on a :

```
def jump(self):
    self.vely = -25 # Modifie le vecteur vitesse de l'oiseau de -25 (Négatif car vers le haut)

def affectGravity(self):
    self.vely += Bird.g # Loi de Newton : la gravité est simulé en affectant une constante au vecteur vitesse
    self.rect.y += self.vely # On applique le vecteur vitesse à la position de l'oiseau
```

La fonction qui permet de gérer les Tuyaux:

```
for pipe in self.list_pipes: # Itère dans la liste de tous les Pipes
    pipe.update() # Met à jour la position X des Pipes en utilsant leur méthode correspondante
    if self.bird.collide(pipe): # Si la méthode collide de Bird renvoie True = L'oiseau entre en contact avec le Pipe
        self.launch = False
        self.game = False
        self.gameOverScreen() # Arrete toutes les variables et lance l'écran de fin
```

```
self.bird.affectGravity() # On affecte la gravité à l'oiseau
if self.list_pipes[-1].rect.x <= (2*self.w)//5: # Si le dernier Pipe a passé l'oiseau
    self.list_pipes.append(Pipe()) # On créé un nouveau Pipe qui sera généré tout à droite

if self.list_pipes[0].rect.x <= -self.list_pipes[0].w: # Si le Pipe sort de l'écran
    self.list_pipes.pop(0) # Supprime le Pipe de la liste
```

III. L'intelligence artificielle

A. Déroulement du script

L'implémentation d'un algorithme génétique dans le jeu a été grandement simplifié grâce à une librairie appelée NEAT, celle ci permet de faire des générations de population et de faire

reproduire les oiseaux les plus performants pour les faire évoluer.
L'idée derrière est simple, premièrement le script est configuré de manière à choisir la taille de chaque population (ici 10 oiseaux par générations), dans ce fichier *config-feedforward.txt*, nous pouvons indiquer aussi d'autres éléments, plus ou moins important comme le taux de mutation et comment les oiseaux interprètent la data.

Le script se déroule ensuite ainsi:
Une seule et même fonction, qui est la méthode *runBot()* de la classe Game se lance à chaque nouvelle génération, celle ci génère la population avec toutes les mutations qu'elle implique. Une fois tous les oiseaux lancés dans la partie, ceux si lisent les infos qui leurs importent et décident de si il faut sauter ou pas.
Une fois tous les oiseaux morts, grâce à la libraire NEAT, les 2 oiseaux avec la meilleure fitness « s'accouplent » pour générer une nouvelle population qui est supposée être plus intelligente, le processus se répète jusqu'à ce qu'un oiseau atteint un seuil de fitness max, indiqué dans le fichier de config.

B. Fonction fitness

La fitness est une valeur associée à chaque oiseau qui représente sa « performance », plus elle est élevée, mieux l'oiseau est « fort » au jeu.
Dans un jeu de type Flappy Bird, calculer la fitness des oiseaux est assez simple, elle incrémente de 0.01 à chaque boucle, de 5 si les oiseaux passent dans un Pipe, et baisse de 1 dès qu'ils touchent un Pipe.

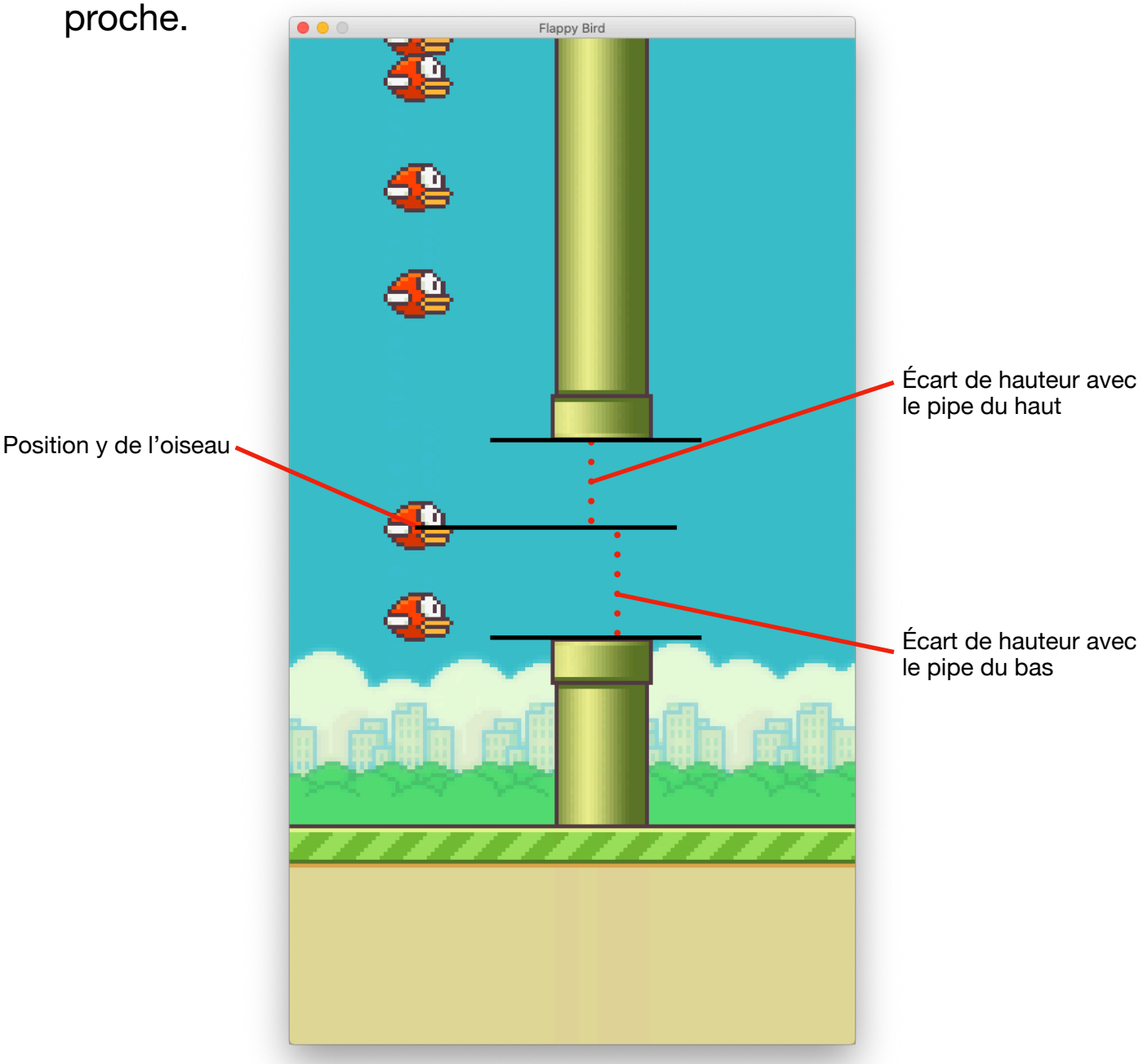
```
ge[x].fitness += 0.01
```

```
# Si les oiseaux encore vivants passent un Pipe
if pipe.rect.x <= bird.rect.x and pipe.rect.x >= bird.rect.x + self.ground.xspeed:
    for g in ge:
        g.fitness += 5 # Augmente de 5 les fitness de tous les oiseaux encore vivants
```

```
for i, bird in enumerate(birds): # pour tous les oiseaux encore vivants
    if bird.collide(pipe): # Si l'oiseau percute un Pipe
        ge[i].fitness -= 1 # Décrémente de 1 sa fitness
        birds.pop(i) # Retire l'oiseau des oiseaux restants
```

C. Interaction des oiseaux avec le jeu

Chaque oiseau possède ce qu'on appelle des gènes, c'est ce qui diffère d'un oiseau à l'autre.
Dans le cas d'un Flappy Bird, chaque oiseau reçoit 3 valeurs : la position y de l'oiseau, la différence de y avec le Pipe haut le plus proche, et la différence de y avec le Pipe bas le plus proche.



Avec ces 3 valeurs les gènes renvoient une seule valeur, comprise entre -1 et 1. Si cette valeur est supérieure à 0.5, alors l'oiseau saute, sinon, il ne fait rien.

```
# ici output est la valeur retournée par les gènes
output = nets[x].activate(
    (bird.rect.y, # Hauteur de l'oiseau
     abs(bird.rect.y - (self.list_pipes[pipe_ind].y + 100)),
     abs(bird.rect.y - (self.list_pipes[pipe_ind].y - 100)))
    ) # ^ Differences de hauteur avec le Pipe du Haut et du Bas

# Si la valeur est supérieure à 0.5
if output[0] > 0.5:
    bird.jump() # L'oiseau saute
```

D. Reproduction

La dernière étape de l'évolution est la reproduction, une fois que tous les oiseaux sont morts, NEAT récupère automatiquement les 2 oiseaux qui ont la plus grosse fitness, et fait un mix de leurs gènes, cela permet d'avoir une nouvelle population d'oiseaux qui ont pour base les gènes d'oiseaux qui ont le mieux performé la génération précédente.
Aussi, à la génération d'une nouvelle population, pour éviter d'avoir une population de mêmes oiseaux, il y a un taux de mutation qui permet de modifier les gènes de l'oiseau, cela permet à l'oiseau d'avoir une chance de progresser si jamais aucun avancement n'est fait par leurs parents.